Dans la 1ère partie nous avons vu comment créer un objet et comment l'instancier.

Nous avons utilisé un mot clé "**public**" pour déclarer nos propriétés et nos méthodes.

Ce mot clé permet de définir la **visibilité de nos propriétés et méthodes**, élément indispensable au bon fonctionnement de nos objets et de notre logique, basé sur le principe d'encapsulation.

Une bonne gestion de l'**encapsulation** permet de **protéger l'utilisation de votre objet** par les développeurs qui seraient amenés à en faire usage.

Les différentes visibilités

Il existe 3 visibilités pour nos propriétés et méthodes :

- **public** : la propriété ou la méthode pourront être accédés depuis l'intérieur et l'extérieur de la classe
- **private** : l'accès à la propriété ou la méthode est possible uniquement depuis l'intérieur de la classe
- **protected** : équivalent à private mais accessible également dans les classes héritées

Private et Protected

Nous allons traiter des visibilités "**private**" et "**protected**" simultanément, leur fonctionnement étant identique.

Si nous passons les propriétés "**titulaire**" et "**solde**" en private, il ne sera plus possible de les modifier ni d'y accéder directement.

Ainsi, si notre classe est définie comme ceci

```php
/**
 * Classe correspondant à un compte bancaire
 */
class Compte
{
    /**
     * Titulaire du compte
     * @var string
     */
    private $titulaire;

    /**
     * Solde du compte
     * @var float
     */
    private $solde;

   }
```

Index.php

Nous ne pourrons pas accéder à nos propriétés de cette façon

```php
echo $compte1->solde;
```

Nous aurons une erreur d'accès

```
Fatal error: Uncaught Error: Cannot access private property
Compte::$solde
```

Les constantes

On n'en parle pas énormément mais il arrive souvent d'avoir besoin de valeurs constantes dans nos classes.

Dans ce cas, nous les déclarerons comme ceci

```php
const INTERETS = 0;
```

Pour y accéder, nous devrons utiliser le nom de l'objet, l'instance de classe est inutile pour la lecture des constantes.

```php
Compte::INTERETS
```

Les accesseurs

Afin de pouvoir définir et lire nos propriétés en "**private**", nous allons créer des "**accesseurs**", méthodes permettant d'y accéder.

Les getters

Les "**getters**" sont des méthodes permettant de lire les propriétés privées.

La convention veut que la méthode s'appelle "**getPropriete**" où "**Propriete**" est à remplacer par le nom de la propriété concernée. Nous aurons donc "**getSolde**" et "**getTitulaire**" dans notre cas.

Les méthodes seront crées comme ceci

```php
/**
 * Retourne le titulaire du compte
 * @return string Titulaire du compte
 */
public function getTitulaire() :string
{
    return $this->titulaire;
}

/**
 * Retourne le solde du compte
 * @return float Solde du compte
 */
public function getSolde() :float
{
    return $this->solde;
}
```

Nous pourrons donc accéder, par exemple, au titulaire du compte en écrivant

index.php

```php
$compte1->getTitulaire();
```

Les setters

Les "**setters**" permettent de définir la valeur des propriétés privées. Faire un "**setter**" permet, par exemple, de s'assurer que la valeur stockée dans la propriété est cohérente avec ce qui est prévu.

La convention veut que la méthode s'appelle "**setPropriete**" où "**Propriete**" est à remplacer par le nom de la propriété concernée. Nous aurons donc "**setSolde**" et "**setTitulaire**" dans notre cas.

Nous écrirons les "**setters**" de cette façon

```php
/**
 * Définit le titulaire du compte
 * @param string $titulaire Titulaire du compte
 * @return Compte Compte bancaire
 */
public function setTitulaire(string $titulaire):self
{
    $this->titulaire = $titulaire;

    return $this;
}

/**
 * Définit le solde du compte
 * @param float $solde Solde du compte
 * @return Compte Compte bancaire
 */
public function setSolde(float $solde):self
{
    if($solde >= 0){
        $this->solde = $solde;
    }

    return $this;

}
```

Nous accéderons donc à notre "**setter**" de cette façon

```php
// Ce code fonctionnera
$compte1->setSolde(45);
$compte1->setTitulaire('Benoit');

// Ce code provoquera une erreur
  $compte1->setSolde('Benoit');
```

La méthode __toString

La question ayant été posée lors du live, voici une description de la **méthode magique __toString**.

Cette méthode permet de **définir ce qui sera renvoyé** par notre objet si quelqu'un essaie de l'**afficher au moyen d'un "echo"** par exemple.

Nous retournerons une chaîne de caractères de notre choix de la façon suivante

```php
/**
 * Méthode magique pour la conversion en chaîne de caractères
 * @return string
 */
public function __toString()
{
    return "Vous visualisez le compte de  {$this->titulaire}, le solde est de  {$this->solde} euros";
}
```